University of Toronto
CSC369, Fall 2017

Assignment 3: Matthew Steglinski 999165861

Running the algorithms

Algorithm: OPT

Tracefile: tr-simpleloop.ref (10304 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	74.0974	7635	2669	2619	19	2600
100	74.3401	7660	2644	2544	0	2544
150	74.3401	7660	2644	2544	0	2494
200	74.3401	7660	2644	2544	0	2444

Tracefile: tr-matmul.ref (2887976 traces) [./matmul 100]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	79.6586	2300520	587456	587406	586319	1087
100	96.7876	2795178	92798	92698	91611	1087
150	99.0784	2861361	26615	26465	25378	1087
200	99.3329	2868711	19265	19065	17978	1087

Tracefile: tr-blocked.ref (188664 traces) [./blocked 50 10]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	99.5442	187804	860	810	426	384
100	99.6973	188093	571	471	105	366
150	99.7482	188189	475	325	56	269
200	99.7747	188239	425	225	18	207

Tracefile: tr-grep.ref (58832 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	99.1586	58337	495	445	166	279
100	99.4901	58532	300	200	5	195
150	99.5071	58542	290	140	0	140
200	99.5071	58542	290	90	0	90

Algorithm: LRU

Tracefile: tr-simpleloop.ref (10304 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	72.9911	7521	2783	2733	89	2644
100	73.9325	7618	2686	2586	2	2584
150	73.9519	7620	2684	2534	0	2534
200	73.9519	7620	2684	2484	0	2484

Tracefile: tr-matmul.ref (2887976 traces) [./matmul 100]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	63.9462	1846752	1041224	1041174	1040066	1108
100	65.1501	1881520	1006456	1006356	1005275	1081
150	98.8612	2855089	32887	32737	31656	1081
200	98.8616	2855100	32876	32676	31595	1081

Tracefile: tr-blocked.ref (188664 traces) [./blocked 50 10]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	99.3327	187405	1259	1209	801	408
100	99.4853	187693	971	871	495	376
150	99.6708	188043	621	471	115	356
200	99.7143	188125	539	339	49	290

Tracefile: tr-grep.ref (58832 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	98.7014	58068	764	714	404	310
100	99.2827	58410	422	322	78	244
150	99.4561	58512	320	170	4	166
200	99.4765	58524	308	108	0	108

Algorithm: FIFO

Tracefile: tr-simpleloop.ref (10304 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	71.0404	7320	2984	2934	218	2716
100	73.2337	7546	2758	2658	45	2613
150	73.6219	7586	2718	2568	16	2552
200	73.6995	7594	2710	2510	12	2498

Tracefile: tr-matmul.ref (2887976 traces) [./matmul 100]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	60.9665	1760699	1127277	1127227	1083237	43990
100	62.4808	1804430	1083546	1083446	1061224	22222
150	98.8085	2853566	34410	34260	32943	1317
200	98.8265	2854087	33889	33689	32433	1256

Tracefile: tr-blocked.ref (188664 traces) [./blocked 50 10]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	99.1806	187118	1546	1496	1005	491
100	99.4466	187620	1044	944	548	396
150	99.6030	187915	749	599	225	374
200	99.6512	188006	658	458	115	343

Tracefile: tr-grep.ref (58832 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	97.8311	57556	1276	1226	815	411
100	99.1229	58316	516	416	132	284
150	99.3660	58459	373	223	15	208
200	99.4544	58511	321	121	0	121

Algorithm: CLOCK

Tracefile: tr-simpleloop.ref (10304 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	72.9231	7514	2790	2740	98	2642
100	73.9130	7616	2688	2588	4	2584
150	73.9422	7619	2685	2535	0	2535
200	73.9422	7619	2685	2485	0	2485

Tracefile: tr-matmul.ref (2887976 traces) [./matmul 100]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	63.9461	1846748	1041228	1041178	1040076	1102
100	65.3115	1886181	1001795	1001695	1000614	1081
150	98.7990	2853292	34684	34534	33452	1082
200	98.8612	2855087	32889	32689	31608	1081

Tracefile: tr-blocked.ref (188664 traces) [./blocked 50 10]

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	99.3041	187351	1313	1263	857	406
100	99.4859	187694	970	870	494	376
150	99.6613	188025	639	489	118	371
200	99.6910	188081	583	383	62	321

Tracefile: tr-grep.ref (58832 traces)

Mem. Size	Hit rate	Hit count	Miss count	Overall eviction count	Clean eviction count	Dirty eviction count
50	98.7014	58068	764	714	398	316
100	99.2946	58417	415	315	74	241
150	99.4357	58500	332	182	5	177
200	99.4765	58524	308	108	0	108

Program of my choice

The fourth program that I chose to analyze was the grep command. Grep has a lot of different options possible, but in the end I opted to just do a simple recursive search using "grep -r *". In order to accomplish this I had to modify the runit script by removing the './' character before '$1' in the valgrind command. Once this edit was made, the trace was generated by typing './runit grep -r *'. To get a better understanding of how the trace would change, I ran the command twice: first on an empty directory, and second on the same directory with the first trace in it.

It was interesting to observe that even though the command ran on an empty directory and would have had no output, there were 59360 memory references present in the trace output from runit. This indicated to me that the program has a seemingly heavy startup cost even if it won't be doing anything. Furthermore, on the second run through of the command the output trace only increased by small 1960 lines to 61320. This was a bit of a shocker since I would have expected the footprint to be much larger considering the program had to search nearly 60000 lines in the first file.

The four algorithms

In a general overview, we observe that FIFO always under performs the rest of the algorithms and OPT is at the top. In the middle however, we have a very close race between LRU and CLOCK with LRU having a slight edge over the CLOCK algorithm. OPT is no surprise, since it carries a crystal ball and can see into the future. It is the optimal method and as such when evicting a page, it always picks the best possible candidate that won't be used the longest time. It is interesting to see certain similarities in the results of LRU and CLOCK such as the same values under grep with memory values 50 and 200. These results between LRU and CLOCK do make some sense however, since the CLOCK algorithm essentially sweeps through possible candidates and approximates a good LRU candidate. FIFO performs worse than these two algorithms because it does not consider that a frame may be being used more frequently, which is what LRU and CLOCK take into consideration.

LRU observations

In general, as the memory size for each test of the tracefiles increases, the hit rate for LRU increases. With each memory increase LRU is able to store a larger amount of recently used pages, which gives the algorithm better information when making a replacement decision. LRU performs poorly when there is little temporal locality, so as the memory size increases we can expect LRU to perform better.

As mentioned previously (in the "The four algorithms" section), in comparison to the other algorithms LRU performs better overall (asides from OPT). Since it uses locality it possesses a great advantage over an algorithm such as FIFO which at any given moment only contains a snapshot of the most recently used pages. CLOCK utilizes a similar concept although only roughly estimating a page that hasn't been used recently. LRU has the advantage since it will always choose the page that hasn't been used recently although may encounter (sometimes) a slightly higher performance cost.

When running LRU on the matmul algorithm, there is a noticable jump between a memory size of 100 to 150. Since LRU operates on locality, the jump from 100 pages to 150 pages enables the algorithm to fully store at least 1 full matrix thus significantly reducing the number of evictions (from over 1 million to about 32000). For other algorithms, LRU maintains a very consistent hit rate and only waivers by a few percent between memory jumps.